Caswell Network-Bypass

User Manual

V2.0.0

7F, No. 186, Jian-Yi Rd., Chung-Ho City, 235 Taipei, Taiwan http://www.cas-well.com
T: +886-2-5591-1999 F: +886-2-5591-1666 E-mail: info@cas-well.com

Important Notice

Caswell, Inc. makes no representations about the suitability of the content, material or information contained in this document ("the Document") for any purpose. All content is provided on an "as is" basis without warranty of any kind. Caswell, Inc. hereby disclaims all warranties and conditions with regard to the Document, including all implied warranties and conditions of satisfactory quality, fitness for a particular purpose, title and non-infringement and any other conditions, warranties and other terms which might otherwise be implied by statute, common law or the law of equity.

While we attempt to ensure that the Document is accurate, we cannot guarantee that it will always be fault-free or up to date. The Document could include technical inaccuracies, typographical errors or out of date material. We endeavor to correct errors and omissions as quickly as practicable. We do not accept liability for any such errors and omissions, or any out of date material. Commentary and other materials included in the Document are not intended to amount to advice on which reliance should be placed and we therefore disclaim all liability and responsibility arising from any such reliance.

The Document is for informational and instructional purposes. Caswell, Inc. reserves the right to make changes in the specifications and other information contained in the Document without prior notice, and the reader should, in all cases, consult Caswell, Inc., to determine whether any changes have been made.

Table of Contents:

7F, No. 186, Jian-Yi Rd., Chung-Ho City, 235 Taipei, Taiwan http://www.cas-well.com
T: +886-2-5591-1999 F: +886-2-5591-1666 E-mail: info@cas-well.com

1. Revision History:

Revision	Release Date	Summary
1.0.0	2009/03/19	1. First release.
1.0.1	2009/3/24	1. Add NAR-5520 & NAR-5530 & NAR5650 motherboard support
1.0.2	2009/4/16	1. Add ABN484L support 2. Change I2c Address permutation and combination
1.0.3	2009/4/27	1. Support MIPS-Cavium bypass
1.0.4	2009/5/18	1. Delete Addressing Driver 2. Create Addressing script
1.0.5	2009/6/8	1. Add the following motherboard support: 1.1 X86 series : CAR2000, CAR3000, NAR7100 1.2 Cavium series: CAPK0100 2. Modify Motherboard board configuration parameters from 0x7090 to NAR7090, mapping table please refer Appendix.
1.1.0	2009/6/30	1. Change the driver make procedure
1.2.0	2009/7/30	1. Add kernel 2.4 /proc system file format description.
1.3.0	2009/8/21	1. Add supported motherboard: NAR5630、NAD-2100L、NAR2200.
2.0.0	2010/15	1. Move support list and appendix to 'SUPPORT_LIST_AND_APPENDIX' in the source code of bypass driver (version V1.5.0 or higher)

2. Supporting list:

Please refer to the file 'SUPPORT_LIST_AND_APPENDIX' in bypass driver source code (with version higher than V1.5.0)

3. General Description:

The setup procedure may include I2C addressing shell script and I2C bypass control driver, the I2c addressing shell script redistributes e1000 register value and e1000 inner eeprom value. Only if the target bypass module allows its I2C address to be configured by software setting (seeing the Appendix), we need to run this addressing shell script.

The bypass control driver will provide a kernel control interface (system file) under the /sys/bus/i2c/devices.

**Note: If kernel is 2.4 kernels, the control interface is under the /proc/sys/dev/sensors.*

**Note2: Bypass control driver is based on standard linux I2C interface, please make sure I2C bus driver (i801) is enabled before bypass driver work.*

4. Make Steps:

4.1. Addressing shell script:

About the description of the addressing, please refer to README which under the script/addressing.

4.2. Make Bypass Control Driver:

We have supported the Bypass control driver on Intel X86 and Cavium MIPS two platforms now. You need to do is make with parameter "TARGET=xx". "KSRC=xx" and "CC=xx" for choosing the target platform and cross-compiling tools.

Make X86 steps:

```
# cd casswell_bypass/driver
# make KSRC=/usr/kernel/xxx CC=/usr/locat/tools/xxx-gcc TARGET=x86
```

Make CAVIUM steps:

```
# make KSRC=/usr/kernel/xxx CC=/usr/locat/tools/xxx-gcc
TARGET=mips_cavium
```

5. Installation Steps:

There are two existing types of bypass in our system, one is on board bypass and the other is plugged like add-card. The installation steps would be running the addressing script (only if add-card bypass supported in the target motherboard) and bypass controller driver one by one. When the bypass driver inserts, the controlling driver needs to pass two configuration parameters. The first is ***board=XXX*** which means the board type of mother board and the second one is ***card_conf=XXX*** for plugging add-card configuration

5.1. On Board Bypass:

When driver inserting, the motherboard type parameter must be passed to driver, the form of parameter is as bellowing Table 1 for the example.

Motherboard Type	Input Parameter
NAR7090	NAR7090
NAR5520	NAR5520

Table 1. Motherboard configuration parameter table

**Note: If the motherboard has no add-card support, the addressing shell script doesn't need to be run. We just skip the step and directly insert the controlling driver.*

5.2. Add-Card Addressing:

If the motherboard supports add-card (ABN482 / ABN484/ ABN484L), we need to run the addressing script after any modifications of add-card sequence. Because the addressing script will use the ethtool command, there must be the Intel network driver pre-loading and ethtool utility supported in the running filesystem.

5.2.1. How to Run shell scrip:

Copy addressing.sh in your target system and run the script:

 # sh addressing.sh display

After script probing, it will discover and show the slot information of bypass add-card by slot ordering.

5.2.2. Additional Parameter:

Set eeprom: If you want to write configuration data, please input "set". It will auto write the probing information to eeprom. For example:

 # sh addressing.sh set

Reset default: If you want to write factory default value to eeprom, please enter "default". For example:

 # sh addressing.sh default

5.3. Add-Card Configuration:

The *card_conf* parameter is passed only when the current mother board supported add-card. The parameter needs to match the add-card type which is inserted now. We will list the supporting add-card and its corresponding configuration parameters in the Appendix. On the other side, if the motherboard doesn't support any bypass card plugged, we just need to pass the board model parameter when bypass driver inserted.

5.3.1. Slot Configure Parameter:

Each slot needs to pass a configuration parameter to identify which add-card is inserted at this slot. We define a mapping relation with our bypass add-card and the input parameter *card_conf*. The details of the input parameters of add-on card are described at the file 'SUPPORT_LIST_AND_APPENDIX' in source code. Table 2 is a sample case of the passed value.

Card Type	Input Parameter
None	0
ABN484	1
ABN482	2

Table 2. Bypass add-cards table

5.3.2. Slot Ordering:

In this section, we describe how to set up the slot ordering by an example. NAR-7090 has three slots. Slot A at the far left and slot C at the far right, we need to

7F, No. 186, Jian-Yi Rd., Chung-Ho City, 235 Taipei, Taiwan http://www.cas-well.com
T: +886-2-5591-1999 F: +886-2-5591-1666 E-mail: info@cas-well.com

take care all the parameters of the three slots. For example, if the ABN484 is mounted on slot A and ABN482 is mounted on slot B. The parameters according to the slot ordering would be like ***card_conf*=0x*120***. The third parameter should be 0 since the slot C is not plugged. For more detail, see the illustration of the mapping table described above in Figure 1.

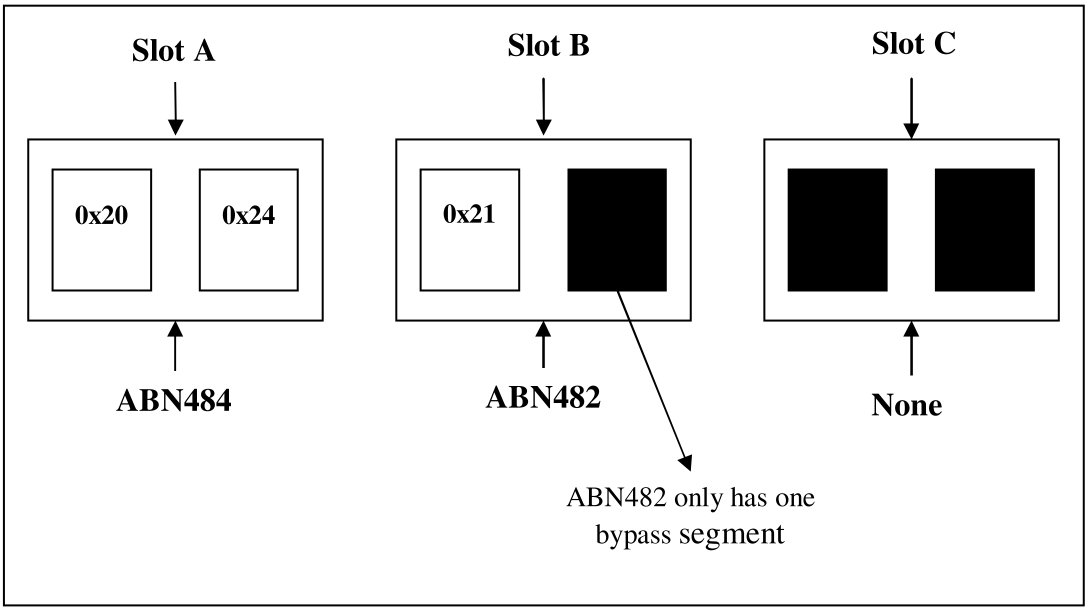

Figure 1. Bypass segments mapping table

5.4. Module Insert:

Example 1: If we control Motherboard NAR7090 with the ABN484L mounted on slot B, the installation steps would be as the following description:

**Note1: If kernel is 2.4 drivers name change to network-bypass.o*

**Note2: the card_conf parameters could be gotten after addressing.*

A. Run Addressing:

 # sh addressing.sh set

Script Logs:

```
Slot=B ETH=eth46 BUSN=03 BYPASSID=4841
Write Data to EEPRROM
Slot=B ETH=eth47 BUSN=04 BYPASSID=4841
Write Data to EEPRROM
……
```

BYPASS Card sequence parameter = 0x030

B. Insert Bypass module:

#insmod network-bypass.ko board=NAR7090 card_conf=0x030

***Example 2*: **If we control Motherboard NAR5650 with the ABN484 mounted on slot A and the ABN482 mounted on slot B, the installation steps would be as the following description:

A. **Run Addressing:**

 # sh addressing.sh set

Script Logs:

……
BYPASS Card sequence parameter = 0x320

B. Insert Bypass module:

#insmod network-bypass.ko board=NAR5650 card_conf=0x320

***Example 3*: **If we configure the motherboard NAR-5520 (No add-card existing), the installation step would be like the following:

Insert Bypass module:

#insmod network-bypass.ko board=NAR5520

6. Add-Card Bypass Address Table:

We arrange a fixed address table for the slots. There is a specific rule to locate the bypass add-card I2C chip after the run addressing script. We use the NAR-7090 as an example; NAR-7090 has 3 bypass add-card slots. The following table 3 showing the relationship with the slot A~C and the add-card type. It means that if the ABN484 is inserted to slot B as an example, the I2C chip address would be fixed at 0x21 and 0x25. If the ABN484 is replaced by ABN482, the I2C chip address would be change to 0x21 since ABN482 only has one bypass segment.

Cards	SlotA(option)	SlotB(option)	SlotC(option)
ABN484	0x20/0x24	0x21/0x25	0x22/0x26
ABN482	0x20	0x21	0x22

Table 3. NAR-7090 Bypass I2C chip address mapping table

<table>
<tr><th>Cards</th><th>SlotA(option)</th><th>SlotB(option)</th><th>SlotC(On Board)</th></tr>
<tr><td>ABN484L</td><td>0x20/0x24</td><td>0x21/0x25</td><td rowspan="2">On Board I2C chip address 0x22/0x26/0x27</td></tr>
<tr><td>ABN482</td><td>0x20</td><td>0x21</td></tr>
</table>

Table 4. NAR-5650 Bypass I2C chip address mapping table

As the table mentioned above, it will create the necessary bypass controlling system files in the corresponding I2C address after the bypass driver loaded. For example, If one ABN484 is inserted into slot C, its address would be 0x22 (segment 1) and 0x26 (segment 2). If we hope to set the bypass mode or get information, we would directly refer to **/sys/bus/i2c/devices/*0x22*** and **/sys/bus/i2c/devices/*0x26.*** Details about the bypass API operation commands please see the section 6.

**Note: If kernel is 2.4 refer folder change to /proc/sys/dev/sensors/pca9555-i2c-0-22 and /proc/sys/dev/sensors/pca9555-i2c-0-26.*

7. Bypass State Diagram:

There are three modes of our bypass controller status. The rough hardware status of each mode is like following figure:

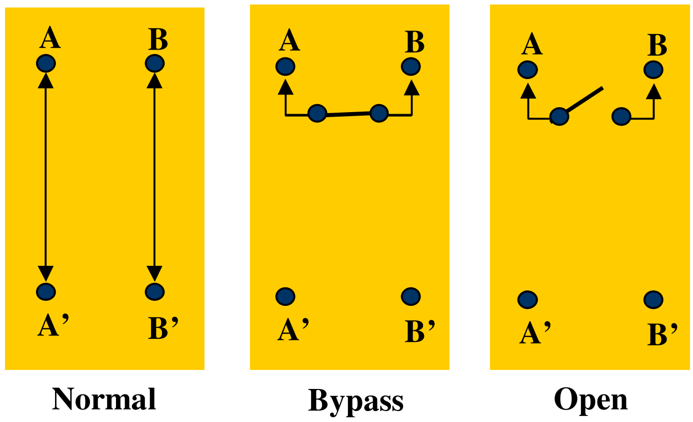

Figure 2. Bypass mode

Normal mode: The packets coming in from one Bypass Ethernet port will be handled by CPU and then be sent out thereafter. It exists only when power is available.

Bypass mode: The packets coming in from one Bypass Ethernet port will be passed to its corresponding port directly without CPU’s intervention. It can occur when either out of power or when power is available.

Open mode: The packets coming in from one Bypass Ethernet port will be dropped. It exists only when system is out of power and BPE is set to “disabled”.

8. Bypass Control Flow:

There are several status and modes migration when a bypass module works. By the mode change, user could reach the goal of network bypass function. The related software API to control the bypass status is described in **Section 9**.

8.1. System status:

When power start up, system will go into normal mode or non-normal mode according to bypass module setting as Figure 3. Which mode system will go into depends on the setting of **NEXTBOOT**

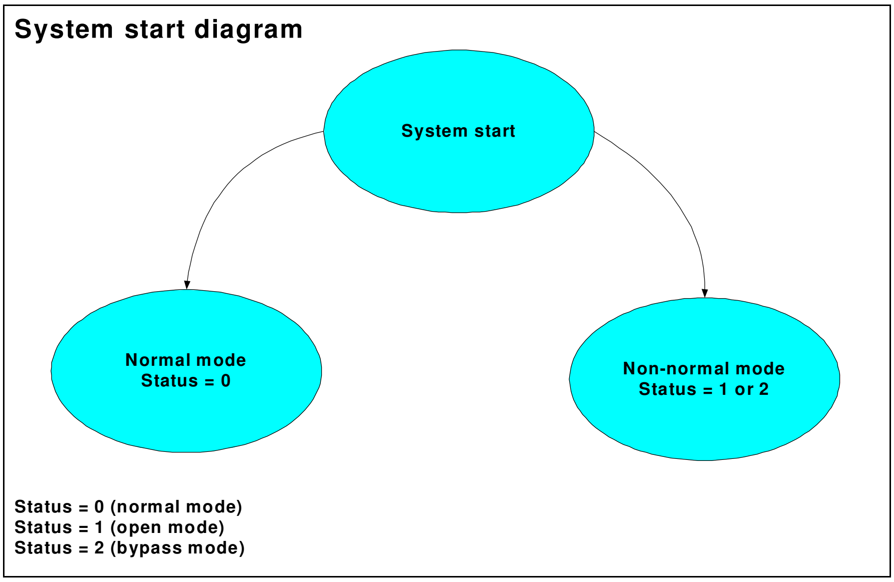

Figure 3. **System start diagram**

8.2. Bypass Mode:

The bypass mode can be changed by software **BYPASS** API, its status flow is as following Figure:

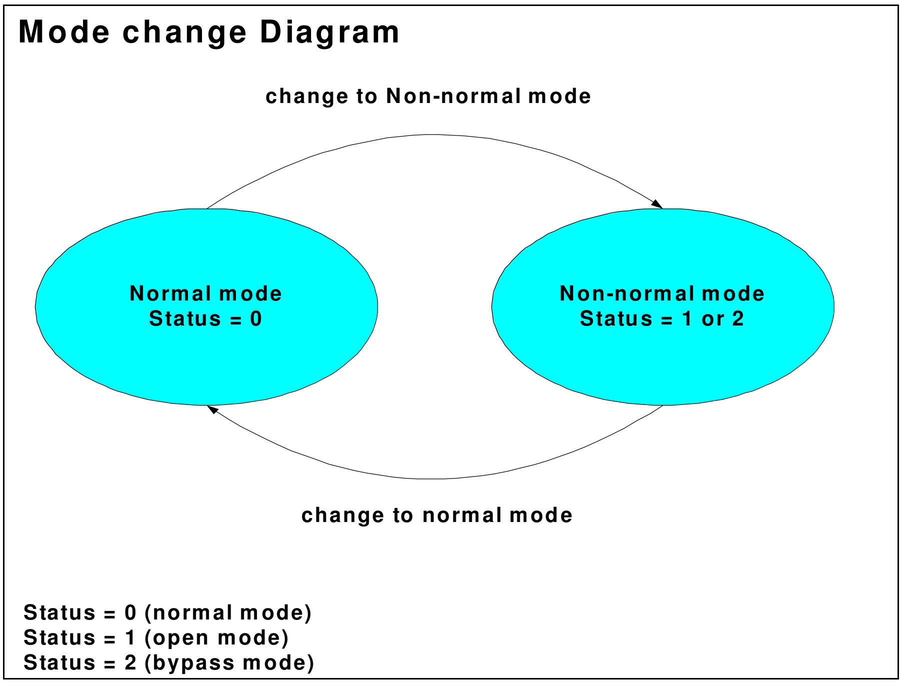

Figure 4. Mode change diagram

8.3. Non-normal mode:

About non-normal mode, it includes two parts, one is open mode, and another is bypass mode. About open mode or bypass mode, they could be set by **BPE** API. The flow is as below:

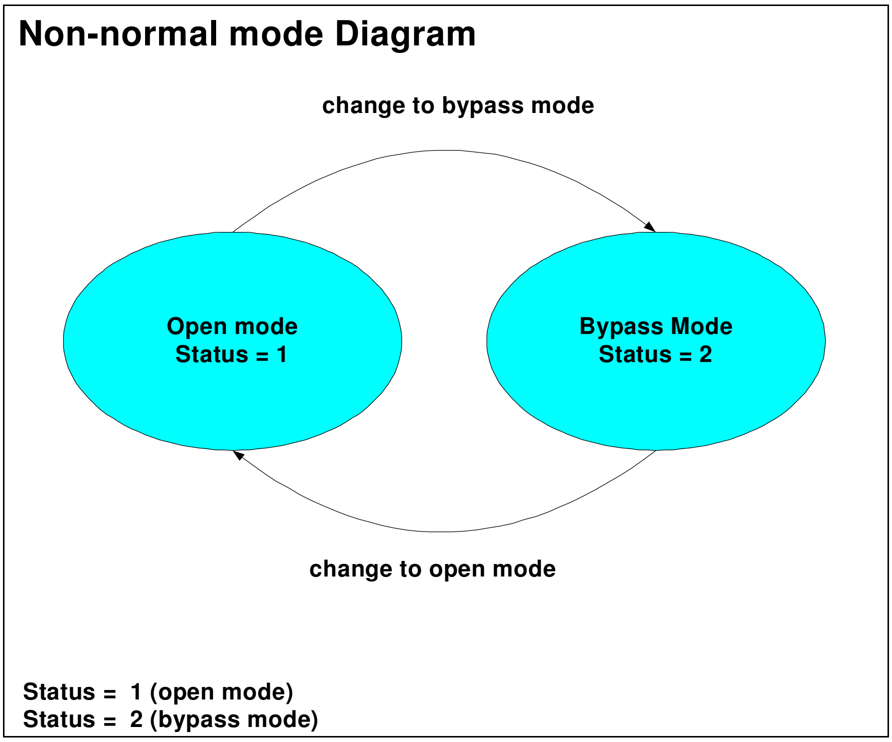

Figure 5. Non-normal mode diagram

8.4. Nextboot setting :

About extboot setting, it is described as below figure 6, it influences bypass status when the system starts up. It can be managed by **Nextboot** API.

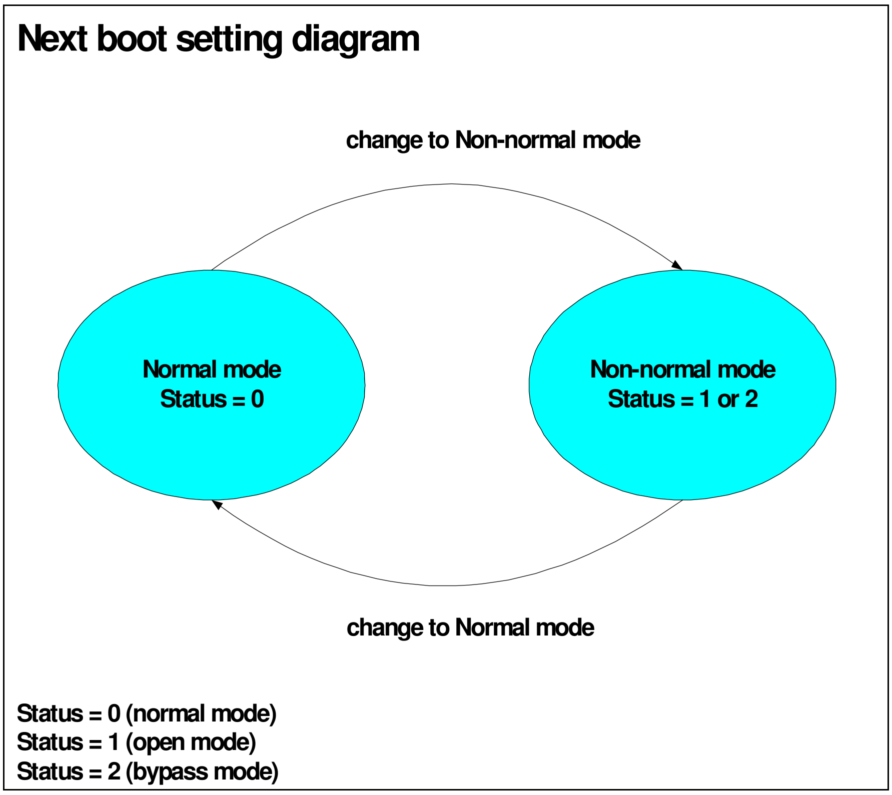

Figure 6. Next boot setting diagram

User can assign system be in normal mode or non-normal when system starts up. Set it by **Nextboot** API.

8.5. Watching Dog Timer:

When watch dog timer is armed, system will be in normal mode, and waiting for watch dog timer expiring, after watch dog timer expire, system will be in non-normal mode. You can set it by **WDT** API. About Watch dog timer status, the detail is described as below:

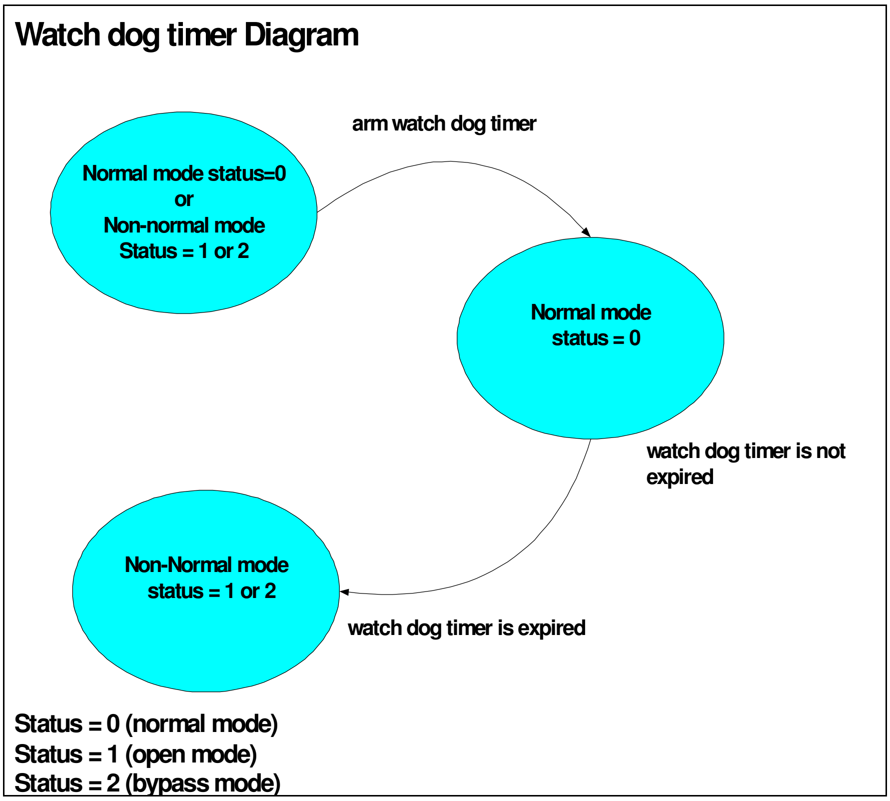

Figure 7. Watch dog timer Diagram

9. Command description:

The following introduce the bypass driver information get/set API, we use system files to handle those. The easiest way is using "echo" and "cat" to do this. Sure, the file open and read/write these system file could be also using. **Notice: The bypass hardware implement has several generations. The default function descriptions would focus on generation 2.** And the hardware generation 1.X has some function differences; we list these as below (Table 5). Also, at each command description note, we list the file operation inconsistence there.

**Note: If kernel is 2.4 refer folder change to /proc/sys/dev/sensors/pca9555-i2c-0-2x.*

Function	GEN1.X
BYPASS	1. Only support normal and non-normal mode changing by software. 2. At non-normal mode, we could change open and bypass mode by hardware jumper setting.
BPE	According to hardware jumper setting.
NEXTBOOT	According to hardware jumper setting.

Table 5: BYPASS version comparison table

9.1. Set BYPASS mode:

Set to Normal:

#echo 0 > /sys/bus/i2c/devices/0-002X/bypass0

Set to Open:

#echo 1 > /sys/bus/i2c/devices/0-002X/bypass0

Set to Bypass:

#echo 2 > /sys/bus/i2c/devices/0-002X/bypass0

**Note: At GEN1.x, we only support normal and non-normal mode change. "echo 2 > …" is invalid and no use.*

9.2. Get BYPASS mode:

#cat /sys/bus/i2c/devices/0-002X/bypass0

=> bypass0 value:

0 => normal, 1=>open, 2=>bypass

**Note: At GEN1.x, we could only get normal (0) and non-normal (1) status. We may need to check hardware jumper to see the setting is open or bypass.*

9.3. Set NEXTBOOT mode:

Set to Normal:

#echo 0 > /sys/bus/i2c/devices/0-002X/nextboot0

Set to Non-normal:

#echo 1 > /sys/bus/i2c/devices/0-002X/nextboot0

**Note: At GEN1.x, it would be according to hardware jumper setting.*

9.4. Get NEXTBOOT mode:

#cat /sys/bus/i2c/devices/0-002X/nextboot0

=> nextboot0 value: 0 => normal, 1=> non-normal

**Note: At GEN1.x, it would be according to hardware jumper setting.*

9.5. Set BPE mode:

Set to Open:

#echo 0 > /sys/bus/i2c/devices/0-002X/bpe0

Set to Bypass:

#echo 1 > /sys/bus/i2c/devices/0-002X/bpe0

**Note: At GEN1.x, it would be according to hardware jumper setting.*

9.6. Get BPE mode:

#cat /sys/bus/i2c/devices/0-002X/bpe0

=> bpe0 value: 0 => open, 1=> bypass

**Note: At GEN1.x, it would be according to hardware jumper setting.*

9.7. Set WDT status:

Disable WDT:

 # echo 0 > /sys/bus/i2c/devices/0-002X/wdt0

Refresh WDT:

#echo 1 > /sys/bus/i2c/devices/0-002X /wdt0

Clear WDT:

 # echo 2 > /sys/bus/i2c/devices/0-002X /wdt0

9.8. Get Time-out status:

#cat /sys/bus/i2c/devices/0-002X /timeout0

=>timeout0 value:

1: WDT time-out occur 0: time-out not occur

9.9. Set WDT period:

Please confirm specification periods are not shared when setting the add-card period. For example, if the ABN484 is inserted to slot B, the I2C chip address would be fixed at 0x21 and 0x25. Because the period is shared, setting to 0x25 is invalid when 0x21 is set to another period value. Note that the watch dog can independently trigger though they share the same period value. (seeing the Appendix).

#echo sec(1~63)>/sys/bus/i2c/devices/0-002X /period0

Note*: In some case, the period value is not corresponding to the unit of second. It has a relation table on Table 5 in the following. The main boards or add-cards listed in appendix with '' are under the rule of this table when their watching dog timer works.*

Value	WDT time-out value
0x01	264.258mS
0x02	528.516mS
0x03	1057.032mS
0x04	2114.065mS
0x05	4228.129mS
0x06	8456.258mS
0x07	16912.52mS
0x08	33825.03mS
0x09	67550.06mS
0x0A	135300.1mS
0x0B	270600.3mS

Table 6: Copper PIC BYPASS WDT period table

9.10. Get WDT period:

#cat /sys/bus/i2c/devices/0-002X /period0

- ⇨ period0 value: period in sec